**Introduction**

My task is the predict the departure delay of a flight between the five busiest airports in the USA. As someone who travels frequently between Chicago O'Hare and Houston International airports, I often run into delays, which made me wonder if there was a way to predict this delay when booking a flight. This project is largely important for the almost 200 million people boarding flights from these airports each year [1]. Knowing in advance whether a flight might be delayed and by how much gives customers the ability to make informed decisions when they fly and avoid the headaches that comes with flight delays.

**Methods**

My data collection was very simple. I was able to find a dataset on the Bureau of Transportation website that contains flight information for all domestic flights operated by major US carriers. To ensure I had a wide range of flight data, I download all of the flights from January 2010 through December of 2017. This dataset, however, had about 48 million flights, which was far too many for me or any model to handle in a reasonable amount of time. I decided to filter for the busiest airports after some testing because the flights from any airport were heavily skewed towards the largest airports. This will eliminate any tail effects. I ran the code in Appendix A to filter by the five busiest airports.

This still left about a million flights, whereas I wanted roughly 200,000 in my training set and another 40,000 in my test set. I ran the code in Appendices B and C to generate 3 different pairs of train and test data. The first is what I call the 'split' data, because it only includes flights that left early/on time and flights that were delayed by more than an hour. The second and third sets are actually the same examples with a different class to predict. The second is the three-class set, which has three different classes: onTime, withinHour, and greaterThanHour. The corresponding third dataset is the same examples with the numeric value for departure delay. These datasets were produced using the C++ code in Appendices A-C.

My experiments are fairly simple. First, I will run a series of classifiers on the split and 3-class data separately. This will include decision trees, 1-nearest neighbor, and neural networks. These will all be compared to ZeroR. I will train linear regression and neural network classifiers on the numeric dataset as well. Because this numeric set has the same examples as the 3-class dataset, I will assign the 3-classes to each numeric prediction classifier to compare its accuracy with that of the 3-class classifiers. All of these algorithms will be run in Weka, and I will perform any post-processing in Excel.

**Results and Analysis**

The results for my tests are presented below in Figures 1-3:

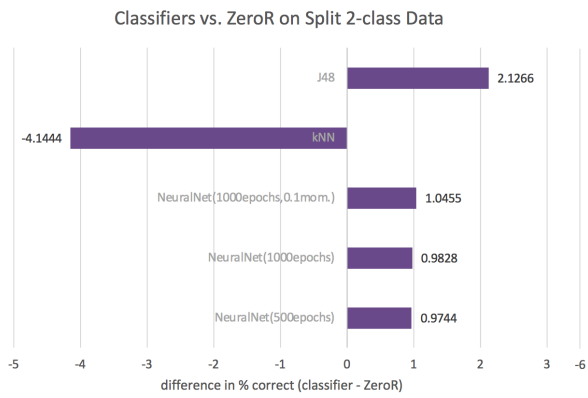


*Figure 1: classifier accuracy on split data*

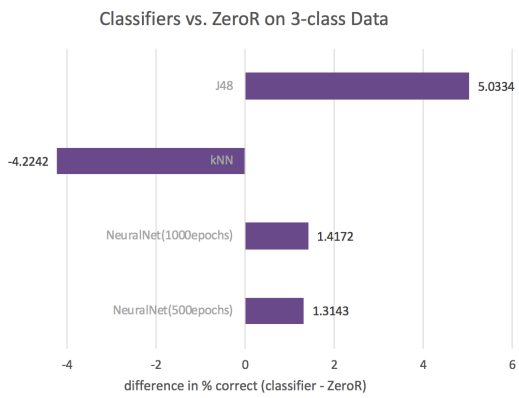


*Figure 2: classifier accuracy on 3-class data*

The first obvious result from our classifiers on discrete classes is that they aren't that much better than ZeroR. For our split data, we improve just 2% and we improve 5% on the 3-class data. This disparity, and the corresponding disparity for the other classifiers makes sense given that ZeroR was around 90% for the split data but only 57% for the 3-class data.

It is interesting to note, however, that the classifiers generally follow the same patterns on the two datasets. Decision trees are best, neural networks are not as good, and nearest neighbor is the worst. This matches my predictions. Nearest neighbor should underperform because so many delayed flights might be on time the next day. The nearest neighbor will often have the wrong class for a delayed flight, so we expect to get many wrong. Neural networks are so powerful that it is not surprising that they perform well. Decision trees perform the best, which was my suspicion all along. It seems logical that certain rules, such as "United flights from ORD to DEN on Friday nights are often delayed" would classify the data well.

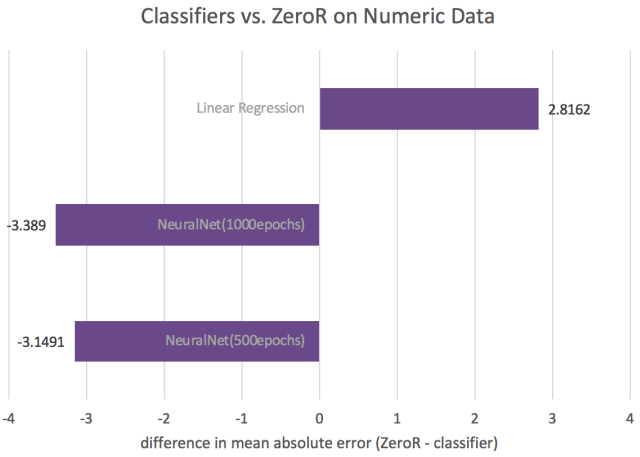


*Figure 3: mean absolute error on numeric data*

Figure 3 above shows our results for the numeric predictors. Both were absolutely horrible. Yes, Linear Regression outperformed ZeroR, but that is an extremely low bar to clear. The most revealing results come from re-classifying the numeric predictions into the three classes. When I did this, the following results came up:

| Linear Regression | | | | Neural Net (500 epochs) | | | |
|---|---|---|---|---|---|---|---|
| | onTime | withinHour | greaterHour | | ontime | withinHour | greaterHour |
| actual | 24864 | 16187 | 2697 | actual | 24864 | 16187 | 2697 |
| pred | 1360 | 42388 | 0 | pred | 254 | 43494 | 0 |
| Neural Net (1000 epochs) | | | | NeuralNet (1000 epoch, 0.1 momentum) | | | |
| | onTIme | withinHour | greaterHour | | onTime | withinHour | greaterHour |
| actual | 24864 | 16187 | 2697 | actual | 24864 | 16187 | 2697 |
| pred | 229 | 43519 | 0 | predicted | 232 | 43516 | 0 |

*Figure 4: classified numeric predictions*

The first thing that jumped out to me was that it never predicted a delay of over an hour. It also predicted very few on time flights. Essentially, we might say it is a very, very cautious predictor: it will almost always tell you your flight will be late, but not necessarily enough of a delay to matter.

To conclude, the highest accuracy seems to be from the decision trees on the classified data, but none of the classifiers were overwhelmingly successful. I suspect I am quite far from true accuracy on this problem, and would need much more time, resources, knowledge, and computational power to truly make accuracy predictions.

**Future Developments**

There are definitely ways to develop this model in the future. To attempt to solve the accuracy problems, it would be great to throw all of the historical data from more than just the 8 years that I used at a Neural Net to see how it would do. Also, it would be best to not have to filter the data because the training sets are too large. This was necessary given the computational limits of my computer.

Additionally, I would like to see what features are weighing more heavily on the models. This would offer insight on what features could be added/dropped/modified to improve accuracy, especially for the numeric predictors.

Finally, I want to understand why the numeric predictors (both Neural Networks and Linear Regression) are so 'cautious' in their predictions. This would give further insight into how to improve these classifiers and hopefully get more accurate predictions.

[1] https://en.wikipedia.org/wiki/List_of_the_busiest_airports_in_the_United_States

## Appendix A: filterTop5.cpp

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <set>
#include <iterator>
#include <ctime>
#include <cstdlib>
using namespace std;

int main()
{
  srand(time(NULL));
  int start_s = clock();
  int linesProcessed = 0;
  const std::string firstLine =
"QUARTER,MONTH,DAY_OF_MONTH,DAY_OF_WEEK,UNIQUE_CARRIER,ORIGIN,DEST,CRS_DEP_TIME,CRS_EL
APSED_TIME,CLASS,TRUE_DELAY";

  // ordered list of the top 15 most popular airports in the USA, by passengers per
year
  const string topports[] = {"ATL", "LAX", "ORD", "DFW", "DEN", "JFK", "SFO", "LAS",
"SEA", "CLT", "MCO", "PHX", "MIA", "EWR", "IAH"};

  const int NUM_AIRPORTS = 5;
  std::stringstream buf;
  ifstream in("unfiltered.csv");
  ofstream out;
  out.open("top5-all.csv");
  if (!in | !out) {
    cout << "Cannot open input or output file." << endl;
  }
  std::string input;
  std::string line;
  std::string token;
  std::string org;
  std::string dst;
  int tn = 0;
  int r, temp = 0, count = 0, trueDelay = 10101;
  int valid = 1;
  out << firstLine << endl;
  while(getline(in, line)) {

    // --------------------------------
    linesProcessed++;
    if (linesProcessed % 100000 == 0)
      cout << "Lines Processed: " << linesProcessed << endl;
    // --------------------------------
    input = line;
    std::istringstream ss(input);
    while (getline(ss, token, ',')) {
      switch(tn) {
      case 0: // quarter
       buf << token << ",";
       break; case 1: // month
       buf << token << ",";
       break; case 2: // DOM
       buf << token << ",";
       break; case 3: // DOW
```

```
   buf << token <<",";
   break; case 4: // carrier
   buf << token <<",";
   break; case 5: // flight num
   break; case 6: // origin
   buf << token <<",";
   org = token;
   break; case 7: // dest
   buf << token <<",";
   dst = token;
   break; case 8: // dep_time
   //map values to the hour (truncate minutes)
   if (token != "") {
     temp = std::stoi(token);
     temp = (temp / 100) * 100;
     buf << temp <<",";
   } else {
     valid = -1;
   }
   break; case 9: // dep_delay
   if (token != "") {
     trueDelay = std::stoi(token);

   }
   break; case 10: // cancelled
   if (std::stoi(token) == 1){
     trueDelay = 9999;
     valid = -1;
   }
   break; case 11: // diverted
   break; case 12: // elapsed_time
   //map values to nearest 10 minutes
   if (token != "") {
     temp = std::stoi(token);
     temp = (temp / 10) * 10;
     buf << temp <<",";
   } else {
     valid = -1;
   }
   break; case 13: // distance

   if (trueDelay == 10101) {
     buf << "unknown";
   } else if (trueDelay > 60) {
     buf << "moreThanHour";
   } else if (trueDelay > 0) {
     buf << "withinHour";
   } else {
     buf << "ontime";
   }
   buf << ","<<trueDelay;


  }
  tn++;
}
int orgcor = 0, dstcor = 0, i = 0;
for (i = 0; i < NUM_AIRPORTS; ++i) {
  if (topports[i] == org)
   orgcor = 1;
  if (topports[i] == dst)
```

```
       dstcor = 1;
       }
    if (valid > 0 && orgcor == 1 && dstcor == 1) {
      out << buf.str() << endl;

      count++;
    }
    buf.clear();
    buf.str(std::string());
    trueDelay = 10101;
    valid = 1;
    tn = 0;
  }
  out.close();

  int stop_s = clock();
  float seconds = (stop_s-start_s)/double(CLOCKS_PER_SEC);
  int hours = 0, mins = 0;
  while (seconds > 3600) {
    hours++;
    seconds -= 3600;
  }
  while (seconds > 60) {
    mins++;
    seconds -= 60;
  }
  cout << "--------------------" << endl;
  cout << "Lines: " << count << endl;
  cout << "--------------------" << endl;
  cout << " Program Run Time" << endl;
  cout << "--------------------" << endl;
  cout << "Hours: " << hours << endl << "Minutes: " << mins << endl << "Seconds: " <<
seconds << endl;
  cout << "--------------------" << endl;
}
```

**Appendix B: filterSplitData.cpp**

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <set>
#include <iterator>
#include <ctime>
#include <cstdlib>
using namespace std;

int main()
{
  srand(time(NULL));
  int start_s = clock();



  ifstream in("top5-all.csv");
  ofstream train;
  ofstream test;
  train.open("top5-split-train.csv");
  test.open("top5-split-test.csv");
  if (!in | !train | !test) {
    cout << "Cannot open one of the files." << endl;
  }
  std::string input;
  std::string line;
  std::string token;
  std::string org;
  std::string dst;
  int countTrain = 0;
  int countTest = 0;
  int linesDone = 0;
  int doWrite;
  int tn = 0;
  while(getline(in, line)) {
    if (linesDone == 0) {
      train << line << endl;
      test << line << endl;

      linesDone++;
    } else {

      input = line;
      std::istringstream ss(input);

      while (getline(ss, token, ',')) {
       if (tn == 9) {
         if (token == "withinHour") {
           doWrite = -1;
         } else {
           doWrite = 1;
         }
       }
       tn++;
      }
      tn = 0;

      linesDone++;
```

```
      if (linesDone % 5000 == 0) {
       cout << "Lines processed: " << linesDone << endl;
      }
      int r = rand() % 100;
      if (r < 30 && doWrite == 1) { // write to train file
       countTrain++;

       train << line << endl;
      } else if (r < 37 && doWrite == 1) { // write to test file
       countTest++;

       test << line << endl;
      }
    }

  }
  test.close();
  train.close();
  int stop_s = clock();
  float seconds = (stop_s-start_s)/double(CLOCKS_PER_SEC);
  int hours = 0, mins = 0;
  while (seconds > 3600) {
    hours++;
    seconds -= 3600;
  }
  while (seconds > 60) {
    mins++;
    seconds -= 60;
  }
  cout << "--------------------" << endl;
  cout << "Lines: " << countTrain << endl;
  cout << "Lines: " << countTest << endl;
  cout << "--------------------" << endl;
  cout << " Program Run Time" << endl;
  cout << "--------------------" << endl;
  cout << "Hours: " << hours << endl << "Minutes: " << mins << endl << "Seconds: " <<
seconds << endl;
  cout << "--------------------" << endl;
}
```

## Appendix C: filterRegressionandClass.cpp

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <set>
#include <iterator>
#include <ctime>
#include <cstdlib>
using namespace std;

int main()
{
  srand(time(NULL));
  int start_s = clock();



  ifstream in("top5-all.csv");
  ofstream regtrain;
  ofstream regtest;
  ofstream classtrain;
  ofstream classtest;
  regtrain.open("top5-reg-train.csv");
  regtest.open("top5-reg-test.csv");
  classtrain.open("top5-class-train.csv");
  classtest.open("top5-class-test.csv");
  if (!in | !regtrain | !regtest | !classtrain | !classtest) {
    cout << "Cannot open one of the files." << endl;
  }
  std::string input;
  std::string line;
  std::string token;
  std::string org;
  std::string dst;
  int countTrain = 0;
  int countTest = 0;
  int linesDone = 0;
  int doWrite;
  int tn = 0;

  while(getline(in, line)) {
    if (linesDone == 0) {
      regtrain << line << endl;
      regtest << line << endl;
      classtrain << line << endl;
      classtest << line << endl;

      linesDone++;
    } else {

      input = line;
      std::istringstream ss(input);
      doWrite = 1;
      linesDone++;
      if (linesDone % 5000 == 0) {
       cout << "Lines processed: " << linesDone << endl;
      }
      int r = rand() % 100;
      if (r < 19 && doWrite == 1) { // write to train file
```

```
        countTrain++;

        regtrain << line << endl;
        classtrain << line << endl;
       } else if (r < 23 && doWrite == 1) { // write to test file
        countTest++;

        regtest << line << endl;
        classtest << line << endl;
       }
     }

   }
   regtest.close();
   regtrain.close();
   classtest.close();
   classtrain.close();
   int stop_s = clock();
   float seconds = (stop_s-start_s)/double(CLOCKS_PER_SEC);
   int hours = 0, mins = 0;
   while (seconds > 3600) {
     hours++;
     seconds -= 3600;
   }
   while (seconds > 60) {
     mins++;
     seconds -= 60;
   }
   cout << "--------------------" << endl;
   cout << "Lines: " << countTrain << endl;
   cout << "Lines: " << countTest << endl;
   cout << "--------------------" << endl;
   cout << " Program Run Time" << endl;
   cout << "--------------------" << endl;
   cout << "Hours: " << hours << endl << "Minutes: " << mins << endl << "Seconds: " <<
seconds << endl;
   cout << "--------------------" << endl;
}
```